

# Clasificación Binaria

Estudiantes de Portugués

Sergio Del Castillo Baranda

4/10/2020

## Carga de los datos y librerías

```
students.csv <- file.path(getwd(), 'student-por.csv')
STUDENTS <- read.csv2(file = students.csv, header = TRUE, sep = ';')
```

```
summary(STUDENTS)
```

```
## school sex age address famsize Pstatus
## GP:423 F:292 Min. :15.00 R:127 GT3:358 A: 61
## MS: 76 M:207 1st Qu.:16.00 U:372 LE3:141 T:438
## Median :16.00
## Mean :16.58
## 3rd Qu.:17.00
## Max. :22.00
## Medu Fedu Mjob Fjob
## Min. :0.000 Min. :0.000 at_home : 93 at_home : 23
## 1st Qu.:2.000 1st Qu.:1.000 health : 41 health : 19
## Median :3.000 Median :2.000 other :198 other :293
## Mean :2.591 Mean :2.385 services:108 services:132
## 3rd Qu.:4.000 3rd Qu.:3.000 teacher : 59 teacher : 32
## Max. :4.000 Max. :4.000
## reason guardian traveltime studytime
## course :209 father:117 Min. :1.000 Min. :1.000
## home :128 mother:351 1st Qu.:1.000 1st Qu.:1.000
## other : 38 other : 31 Median :1.000 Median :2.000
## reputation:124 Mean :1.493 Mean :1.976
## 3rd Qu.:2.000 3rd Qu.:2.000
## Max. :4.000 Max. :4.000
## failures schoolsup famsup paid activities nursery
## Min. :0.0000 no :438 no :178 no :470 no :246 no : 99
## 1st Qu.:0.0000 yes: 61 yes:321 yes: 29 yes:253 yes:400
## Median :0.0000
## Mean :0.1864
## 3rd Qu.:0.0000
## Max. :3.0000
## higher internet romantic famrel freetime
## no : 49 no :103 no :327 Min. :1.00 Min. :1.000
## yes:450 yes:396 yes:172 1st Qu.:4.00 1st Qu.:3.000
## Median :4.00 Median :3.000
## Mean :3.94 Mean :3.198
```

```
##              3rd Qu.:5.00  3rd Qu.:4.000
##              Max.    :5.00  Max.    :5.000
##      goout      Dalc      Walc      health
## Min.    :1.000  Min.    :1.000  Min.    :1.000  Min.    :1.000
## 1st Qu.:2.000  1st Qu.:1.000  1st Qu.:1.000  1st Qu.:2.000
## Median :3.000  Median :1.000  Median :2.000  Median :4.000
## Mean   :3.158  Mean   :1.483  Mean   :2.251  Mean   :3.551
## 3rd Qu.:4.000  3rd Qu.:2.000  3rd Qu.:3.000  3rd Qu.:5.000
## Max.   :5.000  Max.   :5.000  Max.   :5.000  Max.   :5.000
##      absences      G1      G2      G3
## Min.    : 0.000  Min.    : 0.00  Min.    : 0.00  Min.    : 0.00
## 1st Qu.: 0.000  1st Qu.:10.00  1st Qu.:10.00  1st Qu.:11.00
## Median : 2.000  Median :12.00  Median :12.00  Median :12.00
## Mean   : 3.948  Mean   :11.74  Mean   :11.89  Mean   :12.33
## 3rd Qu.: 6.000  3rd Qu.:13.50  3rd Qu.:13.00  3rd Qu.:14.00
## Max.   :32.000  Max.   :18.00  Max.   :19.00  Max.   :19.00
```

## SELECCIÓN DE VARIABLES

El objetivo de este apartado es obtener las mejores variables que nos permitan optimizar nuestro modelos. El trabajo lo realizaremos en dos fases, una fase inicial en la que vamos a realizar una limpieza de datos para obtener un dataset con el que podamos generar un modelo y en segundo lugar lo que realizaremos selección de las mejores variables para optimizar nuestro modelo.

### LIMPIEZA DE NA

No realizamos supresión de NA dado que no hay ninguno en el fichero.

```
check.na(STUDENTS)
```

```
##
## There is a total of  0  NAs on this file
## [1] 0
```

Para comenzar a trabajar con las variables vamos a hacer una selección en función del tipo de variable que es, a continuación trabajaremos con las variables de forma diferente en función de la clase de variable que sea.

Lo primero que haremos será la selección de la variable objetivo (higher) y la separamos del dataset. A continuación, haremos una subdivisión de las columnas restantes entre continuas y categóricas almacenando los nombres de las columnas en dos variables.

```
vardep <- "higher"
students.bis <- STUDENTS[,~which(names(STUDENTS) == vardep)]

continuas <- names(select_if(students.bis, is.integer))
categoricas <- names(select_if(students.bis, is.factor))

cat("Nuestra variable objetivo será: ",vardep, "\n\nVariables continuas: ",continuas, "\n\nVariables ca

## Nuestra variable objetivo será:  higher
##
## Variables continuas:  age Medu Fedu traveltime studytime failures famrel freetime goout Dalc Walc he
##
## Variables categoricas:  school sex address famsize Pstatus Mjob Fjob reason guardian schoolsup famsup
```

## CREACIÓN DE VARIABLES DUMMY

Generamos variables dummy a partir de nuestras variables categóricas. En nuestro caso lo realizamos de todas dado que las variables categóricas no contienen un número demasiado elevado de valores diferentes.

```
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored  
  
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts =  
## FALSE): non-list contrasts argument ignored
```

## ESTANDARIZACIÓN DE VARIABLES

A continuación estandarizamos las variables continuas. Para ello realizamos la media y desviación típica de las continuas y a continuación las estandarizamos. Para trabajar ahora con todas las variables como continuas, las uno a las variables dummy generadas en el paso anterior.

```
means <- apply(students.df[,continuas],2,mean)
sds <- sapply(students.df[,continuas],sd)

students.df.bis <- scale(students.df[,continuas], center = means, scale = sds)
numerocont <- which(colnames(students.df) %in% continuidades)
students.df.s <- cbind(students.df.bis, students.df[, -numerocont])
```

## SELECCIÓN DE VARIABLES

El primer paso en la selección de las variables es suprimir de las variables dummy una variable, dado que esta puede ser obtenida como una negación del resto de las variables.

```
continuas <- c("age", "Medu", "Fedu", "traveltime", "studytime", "failures",
"famrel", "freetime", "goout", "Dalc", "Walc", "health", "absences",
"G1", "G2", "G3", "school.GP", "sex.M",
"address.R", "famsize.GT3", "Pstatus.A",
"Mjob.health", "Mjob.other", "Mjob.services",
"Mjob.teacher", "Fjob.health", "Fjob.other",
"Fjob.services", "Fjob.teacher", "reason.course", "reason.home",
"reason.reputation", "guardian.father", "guardian.mother",
"schoolsup.yes",
"famsup.yes", "paid.yes", "activities.yes",
"nursery.yes", "higher", "internet.yes",
"romantic.yes")

categoricas <- c("")

numerocont <- which(colnames(students.df.s) %in% continuidades)
students.df.s <- students.df.s[, numerocont]

students.df.s$higher<-ifelse(students.df.s$higher=="yes", "Yes", "No") # Corrección de los datos para que

cat("Variables continuas: ",continuas, "\n\nVariables categoricas: ",categoricas)

## Variables continuas: age Medu Fedu traveltime studytime failures famrel freetime goout Dalc Walc he
##
## Variables categoricas:
```

## SELECCIÓN DE VARIABLES EN CLASIFICACIÓN BINARIA LOGÍSTICA

Para la selección de variables hacemos la búsqueda mediante el uso de la medida de ajuste AIC. Para ejecutar los algoritmos lo realizaremos mediante el método stepwise que va incluyendo y sacando variables con el objetivo de optimizar la selección.

```
full<-glm(factor(higher)~., data=students.df.s, family = binomial(link="logit"))

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

null<-glm(factor(higher)~1, data=students.df.s, family = binomial(link="logit"))
```

```
seleccion<-stepAIC(null,scope=list(upper=full),direction="both", trace=0)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
variables <- names(seleccion$coefficients)[-1]
```

```
cat("La mejor selección de variables viene dada por: ", variables)
```

```
## La mejor selección de variables viene dada por: G1 age studytime G3 school.GP famsup.yes Mjob.health
```

## GENERACIÓN DE LOS SETS DE DATOS (train, test / Validación cruzada)

En el anterior apartado hemos obtenido las mejores variables para poder generar nuestros modelos. En este apartado lo que vamos a realizar es una división de los datos en dos sets, uno para la parte de test y otro para la parte de entrenamiento del modelo. El objetivo es utilizar el set de entrenamiento para entrenar nuestro modelo y prepararlo para la predicción y realizar pruebas para comprobar la eficacia con la que es capaz de predecir sobre nuestro set de test.

La validación de los datos la realizaremos mediante validación cruzada que lo que realiza es la selección del mejor conjunto de datos que formarán parte de cada set mediante la comprobación redundante de diferentes escenarios de manera que los datos que queden en un set y otro estén lo más balanceados posible.

Utilizaremos validación cruzada repetida dado que únicamente tenemos un set de 500 filas de datos. La generación de los sets de train y test se realiza 4 veces

```
set.seed(1234)
```

```
control<-trainControl(method = "repeatedcv",number=4,savePredictions = "all")
```

## COMPARACIÓN DE MODELOS

### MODELO CON REGRESIÓN LINEAL

Modelo con regresión lineal, este no tendrá rejilla porque no tiene hiperparámetros.

```
reg<- train(factor(higher)~G1+age+studytime+G3+school.GP+famsup.yes+Mjob.health+schoolsup.yes+Walc+famr
          data=students.df.s,
          method="glm",
          trControl=control,
          trace=FALSE)
```

```
reg
```

```
## Generalized Linear Model
```

```
##
```

```
## 499 samples
```

```
## 11 predictor
```

```
## 2 classes: 'No', 'Yes'
```

```
##
```

```
## No pre-processing
```

```
## Resampling: Cross-Validated (4 fold, repeated 1 times)
```

```
## Summary of sample sizes: 375, 375, 373, 374
```

```
## Resampling results:
##
##   Accuracy   Kappa
##   0.9097716  0.3866103
```

## MODELO CON RED NEURONAL

Ahora vamos a generar un modelo con redes neuronales. Para comprobar su eficacia realizaremos diferentes tuneos hasta obtener el mejor resultado. La forma que tenemos de realizar el tuneado mediante el uso de una rejilla.

```
nnetgrid <- expand.grid(size=c(1,2,3,5,10),
                        decay=c(0.01,0.1,0.001),
                        bag=FALSE)

rednnet<- train(factor(higher)~G1+age+studytime+G3+school.GP+famsup.yes+Mjob.health+schoolsup.yes+Walc+
                data=students.df.s,
                method="avNNet",linout = FALSE,
                maxit=100,
                trControl=control,
                tuneGrid=nnetgrid,
                repeats=5,
                verbose=FALSE,
                trace=FALSE)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
rednnet
```

```
## Model Averaged Neural Network
##
## 499 samples
## 11 predictor
## 2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold, repeated 1 times)
## Summary of sample sizes: 374, 374, 374, 375
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0.001  0.8957581  0.3806485
##   1     0.010  0.9017419  0.3902573
##   1     0.100  0.9017742  0.3336351
##   2     0.001  0.9097903  0.4177093
##   2     0.010  0.8977419  0.3612457
##   2     0.100  0.9138065  0.4318188
##   3     0.001  0.9117903  0.4324913
##   3     0.010  0.9138065  0.4289713
##   3     0.100  0.9138226  0.4295853
##   5     0.001  0.9057903  0.4097755
##   5     0.010  0.9017903  0.3905407
##   5     0.100  0.9058065  0.3998800
##   10    0.001  0.9037903  0.4311235
##   10    0.010  0.9017903  0.3923020
##   10    0.100  0.9037742  0.3743618
```

```
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3, decay = 0.1 and bag
## = FALSE.

bestTuneNnet <- function(nnetmodel, size=FALSE, decay=FALSE){
  # Función que ayuda a obtener el mejor resultado obtenido en un modelo NEURAL NET
  bestSize <- rednnet$bestTune$size
  bestDecay <- rednnet$bestTune$decay
  # Cojo los parámetros de la función si están establecidos
  if (size != FALSE) {bestSize <- size}
  if (decay != FALSE) {bestDecay <- decay}
  nnetmodel$results[nnetmodel$results$size == bestSize &
    nnetmodel$results$decay == bestDecay,]
}
```

## RANDOM FOREST

```
set.seed(1234)
rfgrid<-expand.grid(mtry=c(2,3,4,5,6))

rf<- train(factor(higher)~G1+age+studytime+G3+school.GP+famsup.yes+Mjob.health+schoolsup.yes+Walc+famre.
  data=students.df.s,
  method="rf",
  trControl=control,
  tuneGrid=rfgrid,
  linout = FALSE, ntree=300, nodesize=10,
  replace=TRUE,
  importance=TRUE,
  trace=FALSE)

rf
```

```
## Random Forest
##
## 499 samples
## 11 predictor
## 2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold, repeated 1 times)
## Summary of sample sizes: 375, 375, 373, 374
## Resampling results across tuning parameters:
##
##  mtry  Accuracy  Kappa
##  2     0.9077394 0.2616311
##  3     0.9117394 0.3259238
##  4     0.9077711 0.3115811
##  5     0.9077550 0.3066177
##  6     0.9097552 0.3184320
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

```
bestTuneRf <- function(rfmodel, mtry=FALSE){
  # Función que ayuda a obtener el mejor resultado obtenido en un modelo RANDOM FOREST
  bMtry <- rfmodel$bestTune$mtry
  # Cojo los parámetros de la función si están establecidos
  if (mtry != FALSE) {bMtry <- mtry}
  rfmodel$results[rfmodel$results$mtry == bMtry,]
}
```

## GRADIENT BOOSTING

```
set.seed(1234)
gbmgrid<-expand.grid(shrinkage=c(0.1,0.05,0.01),
                     n.minobsinnode=c(10,20),
                     n.trees=c(100,500,1000),
                     interaction.depth=c(1,2,3))

gbm<- train(factor(higher)~G1+age+studytime+G3+school.GP+famsup.yes+Mjob.health+schoolsup.yes+Walc+famr
            data=students.df.s,
            method="gbm",
            trControl=control,
            tuneGrid=gbmgrid,
            distribution="bernoulli",
            bag.fraction=1,
            verbose=FALSE)

gbm
```

```
## Stochastic Gradient Boosting
##
## 499 samples
## 11 predictor
## 2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold, repeated 1 times)
## Summary of sample sizes: 375, 375, 373, 374
## Resampling results across tuning parameters:
##
## shrinkage interaction.depth n.minobsinnode n.trees Accuracy
## 0.01      1                10             100    0.9018193
## 0.01      1                10             500    0.9117716
## 0.01      1                10            1000    0.9057071
## 0.01      1                20             100    0.9018193
## 0.01      1                20             500    0.9117716
## 0.01      1                20            1000    0.9077552
## 0.01      2                10             100    0.9018193
## 0.01      2                10             500    0.9117232
## 0.01      2                10            1000    0.9157396
## 0.01      2                20             100    0.9018193
## 0.01      2                20             500    0.9077391
## 0.01      2                20            1000    0.9097552
## 0.01      3                10             100    0.8977709
## 0.01      3                10             500    0.9117555
## 0.01      3                10            1000    0.9077711
```



##	0.01	3	20	100	0.9018193
##	0.01	3	20	500	0.9097552
##	0.01	3	20	1000	0.9077870
##	0.05	1	10	100	0.9097555
##	0.05	1	10	500	0.8977709
##	0.05	1	10	1000	0.8937704
##	0.05	1	20	100	0.9097555
##	0.05	1	20	500	0.9057714
##	0.05	1	20	1000	0.9057552
##	0.05	2	10	100	0.9076910
##	0.05	2	10	500	0.9057391
##	0.05	2	10	1000	0.9037870
##	0.05	2	20	100	0.9097232
##	0.05	2	20	500	0.9138193
##	0.05	2	20	1000	0.9178036
##	0.05	3	10	100	0.9117555
##	0.05	3	10	500	0.8978346
##	0.05	3	10	1000	0.8918661
##	0.05	3	20	100	0.9117714
##	0.05	3	20	500	0.9057867
##	0.05	3	20	1000	0.8997545
##	0.10	1	10	100	0.9037071
##	0.10	1	10	500	0.8937704
##	0.10	1	10	1000	0.8937704
##	0.10	1	20	100	0.9057552
##	0.10	1	20	500	0.9057552
##	0.10	1	20	1000	0.9037870
##	0.10	2	10	100	0.9137555
##	0.10	2	10	500	0.9057711
##	0.10	2	10	1000	0.8997709
##	0.10	2	20	100	0.9077232
##	0.10	2	20	500	0.9178356
##	0.10	2	20	1000	0.9137875
##	0.10	3	10	100	0.9077711
##	0.10	3	10	500	0.8938182
##	0.10	3	10	1000	0.8958502
##	0.10	3	20	100	0.9077550
##	0.10	3	20	500	0.8957384
##	0.10	3	20	1000	0.8977386
##	Kappa				
##	0.00000000				
##	0.24777960				
##	0.26548485				
##	0.00000000				
##	0.24777960				
##	0.28963103				
##	0.00000000				
##	0.36150872				
##	0.41823162				
##	0.00000000				
##	0.32628467				
##	0.38118347				
##	0.05484418				
##	0.39106014				

```

## 0.37230312
## 0.00000000
## 0.37137094
## 0.39314237
## 0.24276725
## 0.28052505
## 0.26237952
## 0.24276725
## 0.34534355
## 0.34735799
## 0.31179177
## 0.35455279
## 0.36301204
## 0.34637987
## 0.45796247
## 0.47371145
## 0.39106014
## 0.34690162
## 0.35915250
## 0.39592632
## 0.43485869
## 0.42520093
## 0.25752167
## 0.26237952
## 0.26237952
## 0.28166785
## 0.34735799
## 0.34691182
## 0.39757327
## 0.36930094
## 0.35123182
## 0.35647889
## 0.47307303
## 0.47079303
## 0.35956264
## 0.37985621
## 0.39969893
## 0.37524279
## 0.38969801
## 0.41997059
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 500,
## interaction.depth = 2, shrinkage = 0.1 and n.minobsinnode = 20.
bestTuneGbm <- function(gbmModel, n.trees=FALSE, shrinkage=FALSE, n.minobsinnode=FALSE, interaction.depth=FALSE) {
  # Función que ayuda a obtener el mejor resultado obtenido en un modelo GRADIENT BOOSTING MACHINE
  bTrees <- gbmModel$bestTune$n.trees
  bShrink <- gbmModel$bestTune$shrinkage
  bMin <- gbmModel$bestTune$n.minobsinnode
  bInt <- gbmModel$bestTune$interaction.depth
  # Cojo los parámetros de la función si están establecidos
  if (n.trees != FALSE) {bTrees <- n.trees}
  if (shrinkage != FALSE) {bShrink <- shrinkage}
}

```

```

if (n.minobsinnode != FALSE) {bMin <- n.minobsinnode}
if (interaction.depth != FALSE) {bInt <- interaction.depth}
#Devuelve el mejor resultado para los parámetros introducidos
gbmModel$results[gbmModel$results$n.trees == bTrees &
                  gbmModel$results$shrinkage == bShrink &
                  gbmModel$results$n.minobsinnode == bMin &
                  gbmModel$results$interaction.depth == bInt,]
}

```

## XGBOOST

```

set.seed(1234)
xgbmgrid<-expand.grid(
  min_child_weight=c(10),
  eta=c(0.1,0.05,0.03,0.01),
  nrounds=c(100,500,1000),
  max_depth=6,gamma=0,colsample_bytree=1,subsample=1)

xgbm<- train(factor(higher)~G1+age+studytime+G3+school.GP+famsup.yes+Mjob.health+schoolsup.yes+Walc+fam
             data=students.df.s,
             method="xgbTree",
             trControl=control,
             tuneGrid=xgbmgrid,
             verbose=FALSE)

xgbm

```

```

## eXtreme Gradient Boosting
##
## 499 samples
## 11 predictor
## 2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold, repeated 1 times)
## Summary of sample sizes: 375, 375, 373, 374
## Resampling results across tuning parameters:
##
##  eta   nrounds  Accuracy   Kappa
##  0.01   100     0.9018193 0.00000000
##  0.01   500     0.9117878 0.22689691
##  0.01  1000     0.9177878 0.34620781
##  0.03   100     0.9018193 0.03148615
##  0.03   500     0.9137716 0.33179843
##  0.03  1000     0.9157878 0.35635381
##  0.05   100     0.9117878 0.22689691
##  0.05   500     0.9157878 0.35635381
##  0.05  1000     0.9157878 0.35635381
##  0.10   100     0.9177878 0.34620781
##  0.10   500     0.9157878 0.35635381
##  0.10  1000     0.9157878 0.35635381
##
## Tuning parameter 'max_depth' was held constant at a value of 6
## 1

```

```

## Tuning parameter 'min_child_weight' was held constant at a value of
## 10
## Tuning parameter 'subsample' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 100, max_depth = 6,
## eta = 0.1, gamma = 0, colsample_bytree = 1, min_child_weight = 10
## and subsample = 1.

bestTuneXgbm <- function(XgbmModel,
                        nrounds = FALSE,
                        max_depth = FALSE,
                        eta = FALSE,
                        gamma = FALSE,
                        colsample_bytree = FALSE,
                        min_child_weight = FALSE,
                        subsample = FALSE){

  # Función que ayuda a obtener el mejor resultado obtenido en un modelo XGBOOST

  bnrounds <- XgbmModel$bestTune$nrounds
  bmax_depth <- XgbmModel$bestTune$max_depth
  beta <- XgbmModel$bestTune$eta
  bgamma <- XgbmModel$bestTune$gamma
  bcolsample_bytree <- XgbmModel$bestTune$colsample_bytree
  bmin_child_weight <- XgbmModel$bestTune$min_child_weight
  bsubsample <- XgbmModel$bestTune$subsample

  # Cojo los parámetros de la función si están establecidos
  if (nrounds != FALSE) { bnrounds <- nrounds }
  if (max_depth != FALSE) { bmax_depth <- max_depth }
  if (eta != FALSE) { beta <- eta }
  if (gamma != FALSE) { bgamma <- gamma }
  if (colsample_bytree != FALSE) { bcolsample_bytree <- colsample_bytree }
  if (min_child_weight != FALSE) { bmin_child_weight <- min_child_weight }
  if (subsample != FALSE) { bsubsample <- subsample }

  #Devuelve el mejor resultado para los parámetros introducidos
  XgbmModel$results[XgbmModel$results$nrounds == bnrounds &
                    XgbmModel$results$max_depth == bmax_depth &
                    XgbmModel$results$eta == beta &
                    XgbmModel$results$gamma == bgamma &
                    XgbmModel$results$colsample_bytree == bcolsample_bytree &
                    XgbmModel$results$min_child_weight == bmin_child_weight &
                    XgbmModel$results$subsample == bsubsample,]
}

```

Realizamos una comparativa de la precisión todos los modelos anteriores

```

nnetune <- bestTuneNnet(rednnet)
rftune <- bestTuneRf(rf)
gbmtune <- bestTuneGbm(gbm)
xgbmtune <- bestTuneXgbm(xgbm)

models = c(reg$method, rednnet$method, rf$method, gbm$method, xgbm$method)
accuracies = c(reg$results$Accuracy, nnetune$Accuracy, rftune$Accuracy, gbmtune$Accuracy, xgbmtune$Accuracy)

```

```
comparation <- data.frame("Model" = models, "Accuracy" = accuracies)
comparation[order(comparation$Accuracy, decreasing = TRUE),]
```

```
##      Model  Accuracy
## 4      gbm 0.9178356
## 5 xgbTree 0.9177878
## 2  avNNet 0.9138226
## 3      rf 0.9117394
## 1      glm 0.9097716
```

**VOY POR AQUÍ, FALTA SEGUIR AJUSTANDO LOS MODELOS PARA METER LOS TUNEOS OBTENIDOS EN EL SIGUIENTE APARTADO**

## PREPARACIÓN DE MODELOS PARA ENSAMBLADO

```
source ("library/cruzadas avnnet y log binaria.R")
```

```
## -----
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## -----
##
## Attaching package: 'plyr'
##
## The following objects are masked from 'package:dplyr':
##
##      arrange, count, desc, failwith, id, mutate, rename, summarise,
##      summarize
##
## Attaching package: 'reshape'
##
## The following object is masked from 'package:dplyr':
##
##      rename
##
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##      cov, smooth, var
source ("library/cruzada arbolbin.R")
## -----
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```

## -----
##
## Attaching package: 'plyr'
## The following objects are masked from 'package:reshape':
##
##   rename, round_any
## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize
source ("library/cruzada rf binaria.R")

## -----
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## -----
##
## Attaching package: 'plyr'
## The following objects are masked from 'package:reshape':
##
##   rename, round_any
## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize
source ("library/cruzada gbm binaria.R")

## -----
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## -----
##
## Attaching package: 'plyr'
## The following objects are masked from 'package:reshape':
##
##   rename, round_any
## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize
source ("library/cruzada xgboost binaria.R")

## -----

```

```

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## -----

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:reshape':
##
##   rename, round_any

## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize
source ("library/cruzada SVM binaria lineal.R")

## -----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## -----

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:reshape':
##
##   rename, round_any

## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize
source ("library/cruzada SVM binaria polinomial.R")

## -----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## -----

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:reshape':
##
##   rename, round_any

## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize

```

```

source ("library/cruzada SVM binaria RBF.R")

## -----
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## -----
##
## Attaching package: 'plyr'
##
## The following objects are masked from 'package:reshape':
##
##   rename, round_any
##
## The following objects are masked from 'package:dplyr':
##
##   arrange, count, desc, failwith, id, mutate, rename, summarise,
##   summarize
logi<-cruzadalogistica(data=students.df.s,
  vardep=vardep,listconti=variables,
  listclass=c(""), grupos=4,sinicio=1234,repe=5)

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
logi$modelo="Logística"

# Tuneamos con los datos obtenidos en el ajuste anterior
avnet<-cruzadaavnnnetbin(data=students.df.s,
  vardep=vardep,listconti=variables,
  listclass=c(""), grupos=4,sinicio=1234,repe=5,
  size=c(nnet tune$size),decay=c(nnet tune$decay))

##
## Fitting Repeat 1
##
## # weights:  44
## initial  value 368.519001
## iter  10 value 48.275743
## iter  20 value 42.026844

```



```

## iter 30 value 40.314565
## iter 40 value 39.741220
## iter 50 value 38.245263
## iter 60 value 37.951367
## iter 70 value 37.950235
## final value 37.949996
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 202.396507
## iter 10 value 68.101089
## iter 20 value 42.273178
## iter 30 value 39.756867
## iter 40 value 38.525904
## iter 50 value 38.468148
## final value 38.467955
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 217.990017
## iter 10 value 72.704793
## iter 20 value 58.117095
## iter 30 value 42.373284
## iter 40 value 40.003631
## iter 50 value 39.561184
## iter 60 value 38.763794
## iter 70 value 38.591458
## iter 80 value 38.588217
## final value 38.588182
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 161.044969
## iter 10 value 55.190403
## iter 20 value 44.628570
## iter 30 value 43.790751
## iter 40 value 43.072880
## iter 50 value 43.055732
## final value 43.055694
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 125.054152
## iter 10 value 59.692586
## iter 20 value 39.860778
## iter 30 value 38.778177

```

```

## iter 40 value 38.651633
## iter 50 value 38.445105
## iter 60 value 38.441417
## iter 70 value 38.438713
## final value 38.438712
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 133.314501
## iter 10 value 73.097599
## iter 20 value 43.704718
## iter 30 value 35.651606
## iter 40 value 35.133932
## iter 50 value 35.108062
## final value 35.108050
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 108.358616
## iter 10 value 47.610433
## iter 20 value 35.621885
## iter 30 value 34.603311
## iter 40 value 34.507487
## final value 34.507135
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 170.987871
## iter 10 value 64.913467
## iter 20 value 38.006503
## iter 30 value 34.905413
## iter 40 value 34.527510
## iter 50 value 34.507172
## final value 34.507135
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 220.379344
## iter 10 value 63.901486
## iter 20 value 38.403029
## iter 30 value 36.613940
## iter 40 value 35.133433
## iter 50 value 34.814185
## iter 60 value 34.808009
## final value 34.807085
## converged

```

```

##
## Fitting Repeat 5
##
## # weights: 44
## initial value 193.739267
## iter 10 value 73.773703
## iter 20 value 55.462912
## iter 30 value 41.353741
## iter 40 value 36.272809
## iter 50 value 35.583617
## iter 60 value 35.364089
## iter 70 value 34.631001
## iter 80 value 34.471974
## final value 34.471654
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 292.284920
## iter 10 value 76.770934
## iter 20 value 44.540103
## iter 30 value 35.575457
## iter 40 value 34.282461
## iter 50 value 33.925710
## iter 60 value 33.908519
## final value 33.908510
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 130.044664
## iter 10 value 66.900692
## iter 20 value 36.357880
## iter 30 value 34.022521
## iter 40 value 33.922237
## iter 50 value 33.918965
## final value 33.918964
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 167.044258
## iter 10 value 55.984662
## iter 20 value 37.123074
## iter 30 value 34.339731
## iter 40 value 33.944075
## iter 50 value 33.835812
## iter 60 value 33.829106
## final value 33.829102
## converged
##

```

```

## Fitting Repeat 4
##
## # weights: 44
## initial value 142.979664
## iter 10 value 45.559199
## iter 20 value 33.784041
## iter 30 value 33.379465
## iter 40 value 33.278257
## final value 33.277945
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 174.847841
## iter 10 value 52.808989
## iter 20 value 38.961498
## iter 30 value 34.470393
## iter 40 value 33.907608
## iter 50 value 33.833501
## iter 60 value 33.828495
## final value 33.828444
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 208.245686
## iter 10 value 54.421285
## iter 20 value 44.841333
## iter 30 value 42.005638
## iter 40 value 40.883715
## iter 50 value 40.816598
## iter 60 value 40.815895
## iter 60 value 40.815895
## iter 60 value 40.815895
## final value 40.815895
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 287.922475
## iter 10 value 53.094174
## iter 20 value 42.448478
## iter 30 value 40.701596
## iter 40 value 40.551657
## iter 50 value 40.550178
## iter 50 value 40.550177
## iter 50 value 40.550177
## final value 40.550177
## converged
##
## Fitting Repeat 3

```

```

##
## # weights:  44
## initial  value 307.277926
## iter   10 value 57.334636
## iter   20 value 45.456046
## iter   30 value 41.871945
## iter   40 value 41.111021
## iter   50 value 40.354103
## iter   60 value 40.326485
## final   value 40.326338
## converged
##
## Fitting Repeat 4
##
## # weights:  44
## initial  value 193.209892
## iter   10 value 76.880351
## iter   20 value 46.427051
## iter   30 value 42.245951
## iter   40 value 41.228388
## iter   50 value 40.898395
## iter   60 value 40.850872
## iter   70 value 40.835068
## final   value 40.835066
## converged
##
## Fitting Repeat 5
##
## # weights:  44
## initial  value 167.839877
## iter   10 value 58.674069
## iter   20 value 41.694662
## iter   30 value 40.855548
## iter   40 value 40.835194
## final   value 40.835066
## converged
##
## Fitting Repeat 1
##
## # weights:  44
## initial  value 101.268287
## iter   10 value 61.384358
## iter   20 value 43.701537
## iter   30 value 41.584426
## iter   40 value 41.430559
## iter   50 value 39.780839
## iter   60 value 39.612234
## iter   70 value 39.515215
## iter   80 value 39.504466
## final   value 39.504464
## converged
##
## Fitting Repeat 2
##

```

```

## # weights: 44
## initial value 220.928137
## iter 10 value 55.692548
## iter 20 value 42.193870
## iter 30 value 40.724444
## iter 40 value 40.226143
## iter 50 value 40.214315
## final value 40.214247
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 121.672405
## iter 10 value 52.998853
## iter 20 value 41.121048
## iter 30 value 39.694372
## iter 40 value 39.654604
## final value 39.654236
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 203.480939
## iter 10 value 72.402183
## iter 20 value 47.295042
## iter 30 value 42.469585
## iter 40 value 39.823239
## iter 50 value 39.556494
## iter 60 value 39.521623
## final value 39.521523
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 254.333903
## iter 10 value 62.614258
## iter 20 value 43.529838
## iter 30 value 39.885250
## iter 40 value 39.533268
## iter 50 value 39.504524
## final value 39.504463
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 204.282269
## iter 10 value 58.778121
## iter 20 value 38.542330
## iter 30 value 36.297642
## iter 40 value 35.762580

```

```

## iter 50 value 35.752083
## final value 35.752059
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 327.331106
## iter 10 value 55.059176
## iter 20 value 37.361441
## iter 30 value 35.973586
## iter 40 value 35.696776
## iter 50 value 35.667940
## iter 60 value 35.643556
## iter 70 value 35.643286
## final value 35.643285
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 125.757081
## iter 10 value 54.019101
## iter 20 value 38.541837
## iter 30 value 36.099965
## iter 40 value 35.758668
## iter 50 value 35.752128
## final value 35.752059
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 177.569527
## iter 10 value 61.799071
## iter 20 value 42.879850
## iter 30 value 38.126899
## iter 40 value 36.986866
## iter 50 value 36.168846
## iter 60 value 35.766351
## iter 70 value 35.755273
## iter 80 value 35.752084
## final value 35.752059
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 280.868688
## iter 10 value 56.327625
## iter 20 value 38.974785
## iter 30 value 36.409428
## iter 40 value 35.841756
## iter 50 value 35.762884

```

```

## iter 60 value 35.756866
## iter 70 value 35.754450
## iter 80 value 35.752144
## iter 90 value 35.752059
## iter 90 value 35.752059
## iter 90 value 35.752059
## final value 35.752059
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 231.618766
## iter 10 value 55.700972
## iter 20 value 42.141946
## iter 30 value 39.487008
## iter 40 value 38.625938
## iter 50 value 38.346974
## iter 60 value 37.687701
## iter 70 value 37.670569
## final value 37.670509
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 257.689157
## iter 10 value 66.582709
## iter 20 value 42.903137
## iter 30 value 38.921362
## iter 40 value 38.005256
## iter 50 value 37.670955
## iter 60 value 37.635126
## iter 70 value 37.634842
## iter 70 value 37.634842
## iter 70 value 37.634842
## final value 37.634842
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 157.306631
## iter 10 value 65.383307
## iter 20 value 41.342361
## iter 30 value 39.636467
## iter 40 value 38.840501
## iter 50 value 38.626688
## iter 60 value 38.624057
## final value 38.623788
## converged
##
## Fitting Repeat 4
##

```



```

## # weights: 44
## initial value 274.438968
## iter 10 value 52.073776
## iter 20 value 42.849316
## iter 30 value 39.386796
## iter 40 value 37.775710
## iter 50 value 37.670631
## final value 37.670509
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 296.366916
## iter 10 value 59.151807
## iter 20 value 40.328991
## iter 30 value 38.616672
## iter 40 value 38.221005
## iter 50 value 37.962719
## iter 60 value 37.956217
## final value 37.956210
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 300.413510
## iter 10 value 56.411508
## iter 20 value 42.517268
## iter 30 value 38.571796
## iter 40 value 38.390053
## iter 50 value 37.987487
## iter 60 value 37.941428
## iter 70 value 37.910976
## final value 37.910795
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 293.605138
## iter 10 value 58.745229
## iter 20 value 42.539751
## iter 30 value 37.631381
## iter 40 value 37.056072
## iter 50 value 36.905529
## iter 60 value 36.858633
## iter 70 value 36.858433
## final value 36.858432
## converged
##
## Fitting Repeat 3
##
## # weights: 44

```

```

## initial value 273.250190
## iter 10 value 65.664339
## iter 20 value 47.062040
## iter 30 value 38.662624
## iter 40 value 37.071290
## iter 50 value 36.647775
## iter 60 value 36.639645
## final value 36.639643
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 139.758951
## iter 10 value 49.980696
## iter 20 value 37.896185
## iter 30 value 37.059217
## iter 40 value 37.054587
## final value 37.054571
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 149.592706
## iter 10 value 56.773270
## iter 20 value 38.824826
## iter 30 value 37.428304
## iter 40 value 37.208067
## iter 50 value 37.205415
## final value 37.205408
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 202.846035
## iter 10 value 62.384368
## iter 20 value 40.879027
## iter 30 value 39.169553
## iter 40 value 39.104616
## iter 50 value 39.102544
## iter 50 value 39.102543
## iter 50 value 39.102543
## final value 39.102543
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 178.478896
## iter 10 value 61.215340
## iter 20 value 41.451175
## iter 30 value 37.186225

```

```

## iter 40 value 36.914899
## iter 50 value 36.899821
## final value 36.899788
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 236.702694
## iter 10 value 61.962193
## iter 20 value 41.496300
## iter 30 value 38.785384
## iter 40 value 38.126577
## iter 50 value 37.902954
## iter 60 value 37.902466
## iter 60 value 37.902465
## iter 60 value 37.902465
## final value 37.902465
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 230.526424
## iter 10 value 73.635201
## iter 20 value 44.978416
## iter 30 value 40.506838
## iter 40 value 37.756351
## iter 50 value 37.370700
## iter 60 value 37.350367
## final value 37.350336
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 214.918935
## iter 10 value 57.435067
## iter 20 value 41.936412
## iter 30 value 38.079142
## iter 40 value 37.322771
## iter 50 value 37.243659
## iter 60 value 37.228752
## iter 70 value 37.102827
## iter 80 value 37.091508
## final value 37.091461
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 199.573096
## iter 10 value 48.028957
## iter 20 value 37.449940

```

```

## iter 30 value 35.720762
## iter 40 value 35.622246
## iter 50 value 35.611001
## final value 35.610872
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 210.507583
## iter 10 value 55.436016
## iter 20 value 37.443291
## iter 30 value 36.122022
## iter 40 value 35.865610
## iter 50 value 35.635359
## iter 60 value 35.042452
## iter 70 value 35.030912
## final value 35.030896
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 201.043940
## iter 10 value 52.743454
## iter 20 value 38.767042
## iter 30 value 36.144933
## iter 40 value 35.629930
## iter 50 value 35.611407
## final value 35.610871
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 215.039421
## iter 10 value 58.402052
## iter 20 value 43.068259
## iter 30 value 37.292901
## iter 40 value 35.659441
## iter 50 value 35.504809
## iter 60 value 35.320648
## iter 70 value 35.253452
## final value 35.253346
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 132.381771
## iter 10 value 56.681803
## iter 20 value 39.799473
## iter 30 value 36.396583
## iter 40 value 35.061436

```

```

## iter 50 value 34.793739
## iter 60 value 34.785098
## iter 70 value 34.784963
## final value 34.784962
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 109.644873
## iter 10 value 65.786215
## iter 20 value 43.887415
## iter 30 value 38.864788
## iter 40 value 38.341537
## iter 50 value 38.260525
## iter 60 value 38.239096
## iter 70 value 38.210486
## final value 38.209814
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 196.159932
## iter 10 value 55.485149
## iter 20 value 40.433104
## iter 30 value 37.904051
## iter 40 value 37.765763
## iter 50 value 37.756390
## final value 37.756363
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 309.594745
## iter 10 value 56.249811
## iter 20 value 44.530297
## iter 30 value 39.496436
## iter 40 value 38.551614
## iter 50 value 38.196751
## iter 60 value 38.193807
## final value 38.193799
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 167.313779
## iter 10 value 62.756787
## iter 20 value 41.261469
## iter 30 value 38.426951
## iter 40 value 38.380565
## iter 50 value 38.377322

```

```

## final value 38.377320
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 222.469479
## iter 10 value 64.924589
## iter 20 value 46.884130
## iter 30 value 41.017191
## iter 40 value 39.474349
## iter 50 value 38.167021
## iter 60 value 37.914617
## iter 70 value 37.773388
## iter 80 value 37.759055
## iter 90 value 37.757532
## iter 100 value 37.757448
## final value 37.757448
## stopped after 100 iterations
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 129.638510
## iter 10 value 60.585432
## iter 20 value 43.340143
## iter 30 value 38.048164
## iter 40 value 37.583923
## final value 37.583287
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 177.563836
## iter 10 value 57.480195
## iter 20 value 40.296570
## iter 30 value 38.711150
## iter 40 value 38.033099
## iter 50 value 37.777965
## iter 60 value 37.589035
## iter 70 value 37.577727
## iter 80 value 37.563709
## iter 90 value 37.563288
## final value 37.563287
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 234.570447
## iter 10 value 58.801040
## iter 20 value 43.461803
## iter 30 value 40.587630

```

```

## iter 40 value 40.282186
## iter 50 value 40.273623
## iter 60 value 40.243423
## iter 70 value 40.035694
## iter 80 value 39.743748
## iter 90 value 39.730503
## final value 39.726205
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 165.262978
## iter 10 value 67.131976
## iter 20 value 42.172051
## iter 30 value 39.116466
## iter 40 value 38.376242
## iter 50 value 37.564615
## iter 60 value 37.522617
## final value 37.522582
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 119.295008
## iter 10 value 48.964377
## iter 20 value 38.853232
## iter 30 value 38.062117
## iter 40 value 37.844171
## iter 50 value 37.587556
## iter 60 value 37.563314
## final value 37.563287
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 178.601823
## iter 10 value 65.217890
## iter 20 value 41.479777
## iter 30 value 38.060150
## iter 40 value 37.766286
## iter 50 value 37.622193
## iter 60 value 37.431115
## final value 37.430801
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 258.368057
## iter 10 value 62.193473
## iter 20 value 43.232761

```

```

## iter 30 value 39.479234
## iter 40 value 37.737399
## iter 50 value 37.454061
## iter 60 value 37.402743
## final value 37.395450
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 221.450474
## iter 10 value 72.521896
## iter 20 value 41.675449
## iter 30 value 38.343195
## iter 40 value 37.653761
## iter 50 value 37.432273
## iter 60 value 37.430801
## iter 60 value 37.430800
## iter 60 value 37.430800
## final value 37.430800
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 161.019341
## iter 10 value 49.756271
## iter 20 value 38.680331
## iter 30 value 37.672053
## iter 40 value 37.231773
## iter 50 value 37.132718
## iter 60 value 37.117161
## final value 37.117153
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 199.917522
## iter 10 value 58.390353
## iter 20 value 42.333601
## iter 30 value 39.257534
## iter 40 value 38.479657
## iter 50 value 37.398975
## iter 60 value 37.207121
## iter 70 value 37.118321
## final value 37.117153
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 160.855503
## iter 10 value 56.492968

```



```

## iter 20 value 42.320447
## iter 30 value 39.514363
## iter 40 value 38.523879
## iter 50 value 38.396977
## iter 60 value 38.358748
## final value 38.358534
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 166.604782
## iter 10 value 75.410392
## iter 20 value 50.622600
## iter 30 value 45.525869
## iter 40 value 40.265607
## iter 50 value 38.437899
## iter 60 value 38.209160
## iter 70 value 38.202906
## final value 38.202902
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 122.308423
## iter 10 value 58.013790
## iter 20 value 44.478301
## iter 30 value 39.662185
## iter 40 value 38.377253
## iter 50 value 37.759527
## iter 60 value 37.602116
## iter 70 value 37.600324
## final value 37.600314
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 279.097880
## iter 10 value 62.037054
## iter 20 value 40.405247
## iter 30 value 38.141625
## iter 40 value 38.041989
## iter 50 value 38.033638
## final value 38.033634
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 204.123360
## iter 10 value 58.954838
## iter 20 value 41.431020

```

```

## iter 30 value 38.967795
## iter 40 value 38.022337
## iter 50 value 37.616162
## iter 60 value 37.600317
## final value 37.600314
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 242.133450
## iter 10 value 67.937001
## iter 20 value 44.976757
## iter 30 value 37.952719
## iter 40 value 37.671894
## iter 50 value 37.455417
## iter 60 value 37.368937
## iter 70 value 37.357129
## final value 37.356884
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 257.005385
## iter 10 value 57.024461
## iter 20 value 38.818377
## iter 30 value 37.432221
## iter 40 value 37.175394
## iter 50 value 37.163676
## final value 37.163667
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 248.250093
## iter 10 value 60.774450
## iter 20 value 41.310193
## iter 30 value 37.713943
## iter 40 value 37.193151
## iter 50 value 37.163914
## iter 60 value 37.163668
## final value 37.163667
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 256.389914
## iter 10 value 51.441035
## iter 20 value 39.182927
## iter 30 value 37.080585
## iter 40 value 36.151519

```

```

## iter 50 value 36.076566
## iter 60 value 36.042203
## final value 36.042134
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 259.853237
## iter 10 value 48.174908
## iter 20 value 40.432015
## iter 30 value 37.284567
## iter 40 value 37.008806
## iter 50 value 37.005137
## final value 37.005131
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 204.855865
## iter 10 value 65.023167
## iter 20 value 38.102221
## iter 30 value 36.742557
## iter 40 value 36.527184
## iter 50 value 36.502870
## final value 36.502818
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 122.551090
## iter 10 value 63.487240
## iter 20 value 39.842981
## iter 30 value 36.963402
## iter 40 value 36.081063
## iter 50 value 35.910983
## iter 60 value 35.910607
## final value 35.910605
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 166.149555
## iter 10 value 55.333405
## iter 20 value 38.724316
## iter 30 value 37.042319
## iter 40 value 36.322964
## iter 50 value 36.261041
## iter 60 value 36.257949
## final value 36.257935
## converged

```

```

##
## Fitting Repeat 4
##
## # weights: 44
## initial value 191.233778
## iter 10 value 67.840501
## iter 20 value 41.827825
## iter 30 value 36.851395
## iter 40 value 36.203510
## iter 50 value 36.121689
## iter 60 value 36.097358
## final value 36.097292
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 225.505068
## iter 10 value 60.074617
## iter 20 value 39.512483
## iter 30 value 36.681840
## iter 40 value 36.145623
## iter 50 value 36.082538
## iter 60 value 35.979750
## final value 35.979403
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 173.952782
## iter 10 value 57.707819
## iter 20 value 42.287645
## iter 30 value 37.783996
## iter 40 value 37.251320
## iter 50 value 37.085268
## iter 60 value 37.084073
## final value 37.084072
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 236.262185
## iter 10 value 74.865281
## iter 20 value 51.421370
## iter 30 value 38.921194
## iter 40 value 37.335279
## iter 50 value 36.854445
## iter 60 value 36.827489
## final value 36.827317
## converged
##
## Fitting Repeat 3

```

```

##
## # weights:  44
## initial  value 189.929516
## iter   10 value 54.966627
## iter   20 value 39.095089
## iter   30 value 37.921144
## iter   40 value 37.398214
## iter   50 value 37.329172
## iter   60 value 37.327963
## final   value 37.327957
## converged
##
## Fitting Repeat 4
##
## # weights:  44
## initial  value 321.234541
## iter   10 value 52.447297
## iter   20 value 38.698155
## iter   30 value 37.495563
## iter   40 value 37.236034
## iter   50 value 37.234185
## iter   50 value 37.234185
## iter   50 value 37.234185
## final   value 37.234185
## converged
##
## Fitting Repeat 5
##
## # weights:  44
## initial  value 281.102085
## iter   10 value 54.460896
## iter   20 value 43.031712
## iter   30 value 41.792394
## iter   40 value 38.652366
## iter   50 value 37.048226
## iter   60 value 36.869436
## iter   70 value 36.866547
## final   value 36.866546
## converged
##
## Fitting Repeat 1
##
## # weights:  44
## initial  value 212.325687
## iter   10 value 50.348840
## iter   20 value 40.984246
## iter   30 value 39.476377
## iter   40 value 39.417796
## final   value 39.417539
## converged
##
## Fitting Repeat 2
##
## # weights:  44

```

```

## initial value 219.945615
## iter 10 value 50.159213
## iter 20 value 41.372787
## iter 30 value 39.731794
## iter 40 value 39.423915
## iter 50 value 39.417541
## final value 39.417539
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 188.646374
## iter 10 value 58.388483
## iter 20 value 42.872550
## iter 30 value 40.804127
## iter 40 value 40.637774
## iter 50 value 40.632085
## final value 40.632054
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 265.628311
## iter 10 value 68.883317
## iter 20 value 52.547224
## iter 30 value 45.653664
## iter 40 value 41.629003
## iter 50 value 40.055046
## iter 60 value 39.611783
## iter 70 value 39.505734
## iter 80 value 39.503437
## final value 39.503433
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 92.727474
## iter 10 value 50.796500
## iter 20 value 42.269129
## iter 30 value 40.132845
## iter 40 value 39.439794
## iter 50 value 39.367855
## iter 60 value 39.363276
## final value 39.363274
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 227.205324
## iter 10 value 78.822460

```

```

## iter 20 value 46.970799
## iter 30 value 38.710263
## iter 40 value 37.588361
## iter 50 value 37.388525
## iter 60 value 37.372110
## final value 37.372073
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 269.599019
## iter 10 value 60.230695
## iter 20 value 40.980739
## iter 30 value 39.177964
## iter 40 value 38.405826
## iter 50 value 37.534207
## iter 60 value 36.930214
## iter 70 value 36.927168
## final value 36.927164
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 219.145486
## iter 10 value 59.714804
## iter 20 value 40.404807
## iter 30 value 37.793363
## iter 40 value 37.372437
## iter 50 value 37.358325
## final value 37.358319
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 230.381953
## iter 10 value 64.150423
## iter 20 value 39.456021
## iter 30 value 38.171581
## iter 40 value 37.186040
## iter 50 value 36.950933
## iter 60 value 36.928461
## final value 36.928440
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 206.313506
## iter 10 value 53.697232
## iter 20 value 39.732412
## iter 30 value 37.922373

```

```

## iter 40 value 37.627502
## iter 50 value 37.619879
## final value 37.619855
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 272.715787
## iter 10 value 77.426515
## iter 20 value 41.298206
## iter 30 value 35.431990
## iter 40 value 34.945099
## iter 50 value 34.939714
## final value 34.939697
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 263.877048
## iter 10 value 52.931004
## iter 20 value 38.391195
## iter 30 value 35.607039
## iter 40 value 34.793147
## iter 50 value 34.770632
## final value 34.770561
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 173.232712
## iter 10 value 62.272014
## iter 20 value 39.318441
## iter 30 value 35.722354
## iter 40 value 34.977645
## iter 50 value 34.940785
## iter 60 value 34.924364
## iter 70 value 34.499806
## iter 80 value 34.451501
## final value 34.450561
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 239.401265
## iter 10 value 47.904992
## iter 20 value 37.376139
## iter 30 value 36.664210
## iter 40 value 35.313120
## iter 50 value 35.198821
## final value 35.198550

```



```

## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 209.809157
## iter 10 value 59.166713
## iter 20 value 37.255042
## iter 30 value 36.359339
## iter 40 value 35.281771
## iter 50 value 34.799888
## iter 60 value 34.770576
## final value 34.770561
## converged
##
## Fitting Repeat 1
##
## # weights: 44
## initial value 506.046567
## iter 10 value 89.915458
## iter 20 value 64.632778
## iter 30 value 59.568381
## iter 40 value 57.673989
## iter 50 value 51.529777
## iter 60 value 50.240924
## iter 70 value 49.854736
## iter 80 value 49.761762
## iter 90 value 49.761125
## final value 49.761119
## converged
##
## Fitting Repeat 2
##
## # weights: 44
## initial value 163.665474
## iter 10 value 88.337130
## iter 20 value 52.784784
## iter 30 value 49.583276
## iter 40 value 49.459676
## iter 50 value 49.457167
## final value 49.457150
## converged
##
## Fitting Repeat 3
##
## # weights: 44
## initial value 246.670200
## iter 10 value 78.740318
## iter 20 value 56.057565
## iter 30 value 50.316657
## iter 40 value 49.817122
## iter 50 value 49.734862
## iter 60 value 48.951182
## iter 70 value 48.711763

```

```

## iter 80 value 48.441297
## final value 48.440838
## converged
##
## Fitting Repeat 4
##
## # weights: 44
## initial value 305.012654
## iter 10 value 70.087107
## iter 20 value 54.120081
## iter 30 value 51.857717
## iter 40 value 51.013582
## iter 50 value 50.643868
## iter 60 value 50.603459
## iter 70 value 50.579951
## final value 50.579787
## converged
##
## Fitting Repeat 5
##
## # weights: 44
## initial value 336.139689
## iter 10 value 66.752688
## iter 20 value 54.095522
## iter 30 value 50.626735
## iter 40 value 49.890195
## iter 50 value 49.816600
## iter 60 value 49.815571
## iter 60 value 49.815571
## iter 60 value 49.815571
## final value 49.815571
## converged
## size decay bag Accuracy Kappa AccuracySD KappaSD
## 1 3 0.1 FALSE 0.9110182 0.4071574 0.01723245 0.1705323

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

avnet$modelo="AvNet"

arbol<-cruzadaarbolbin(data=students.df.s,

```

```

vardep=vardep,listconti=variables,
listclass=c(""), grupos=4,sinicio=1234,repe=5)

##   cp Accuracy      Kappa AccuracySD   KappaSD
## 1  0 0.8961793 0.2580466 0.01661613 0.1668027

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

arbol$modelo="Árbol"

bag<-cruzadarfbin(data=students.df.s,
vardep=vardep,listconti=variables,
listclass=c(""),
grupos=4,sinicio=1234,repe=5,nodesize=10,
ntree=1000,replace=TRUE,
mtry=rftune$mtry)

##   mtry Accuracy      Kappa AccuracySD   KappaSD
## 1    3 0.9078214 0.2803234 0.01770944 0.175194

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

bag$modelo="Bag"

rf<-cruzadarfbin(data=students.df.s, vardep=vardep,
listconti=variables, listclass=c(""),
grupos=4,sinicio=1234,repe=5,nodesize=10,
mtry=6,ntree=3000,replace=TRUE,sampsize=150)

```

```
## mtry Accuracy Kappa AccuracySD KappaSD
## 1 6 0.9054183 0.294027 0.0191713 0.1786822

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

rf$modelo="RF"

gbm<-cruzadagbmbin(data=students.df.s,
  vardep=vardep,listconti=variables,
  listclass=c(""),
  grupos=4,sinicio=1234,repe=5,
  n.minobsinnode=gbmtune$n.minobsinnode,
  shrinkage=gbmtune$shrinkage,
  n.trees=gbmtune$n.trees,
  interaction.depth=gbmtune$interaction.depth)

## n.minobsinnode shrinkage n.trees interaction.depth Accuracy Kappa
## 1 20 0.1 500 2 0.9134151 0.4373928
## AccuracySD KappaSD
## 1 0.01580442 0.1427231

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

gbm$modelo="gbm"

xgbm<-cruzadaxgbmbin(data=students.df.s,
  vardep=vardep,listconti=variables,
  listclass=c("")),
```

```
grupos=4,sinicio=1234,repe=5,
min_child_weight=xgbmtune$min_child_weight,
eta=xgbmtune$eta,
nrounds=xgbmtune$nrounds,
max_depth=xgbmtune$max_depth,
gamma=xgbmtune$gamma,
colsample_bytree=xgbmtune$colsample_bytree,
subsample=xgbmtune$subsample)
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##   objective
##   Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##  objective
##  Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
##  objective
##  Only the last value for each of them will be used.
```

```
## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
```

```

## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## Warning in check.booster.params(params, ...): The following parameters were provided multiple times:
## objective
## Only the last value for each of them will be used.

## min_child_weight eta nrounds max_depth gamma colsample_bytree subsample
## 1 10 0.1 100 6 0 1 1
## Accuracy Kappa AccuracySD KappaSD
## 1 0.9126055 0.323598 0.01693507 0.1871463

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

```

```

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases
xgbm$modelo="xgbm"

svm<-cruzadaSVMbin(data=students.df.s,
  vardep=vardep,listconti=variables,
  listclass=c(""),
  grupos=4,sinicio=1234,repe=5,C=0.01)

##      C Accuracy      Kappa AccuracySD      KappaSD
## 1 0.01 0.9037924 0.2519177 0.01482343 0.1600169

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

svm$modelo="SVM"

svmp<-cruzadaSVMbinPoly(data=students.df.s,
  vardep=vardep,listconti=variables,
  listclass=c(""),
  grupos=4,sinicio=1234,repe=5,C=2,degree=3,scale=0.1)

##      C degree scale Accuracy      Kappa AccuracySD      KappaSD
## 1 2      3      0.1 0.9086214 0.2005234 0.009123285 0.162453

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

svmp$modelo="SVMPoly"

```

```

svmrbf<-cruzadaSVMbinRBF(data=students.df.s, vardep=vardep,
  listconti=variables,
  listclass=c(""),
  grupos=4,sinicio=1234,repe=5,
  C=1,sigma=0.1)

##   C sigma  Accuracy      Kappa AccuracySD  KappaSD
## 1 1   0.1 0.9142504 0.3902557 0.01775811 0.154883

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

## Setting levels: control = No, case = Yes
## Setting direction: controls < cases

svmrbf$modelo="SVMRBF"

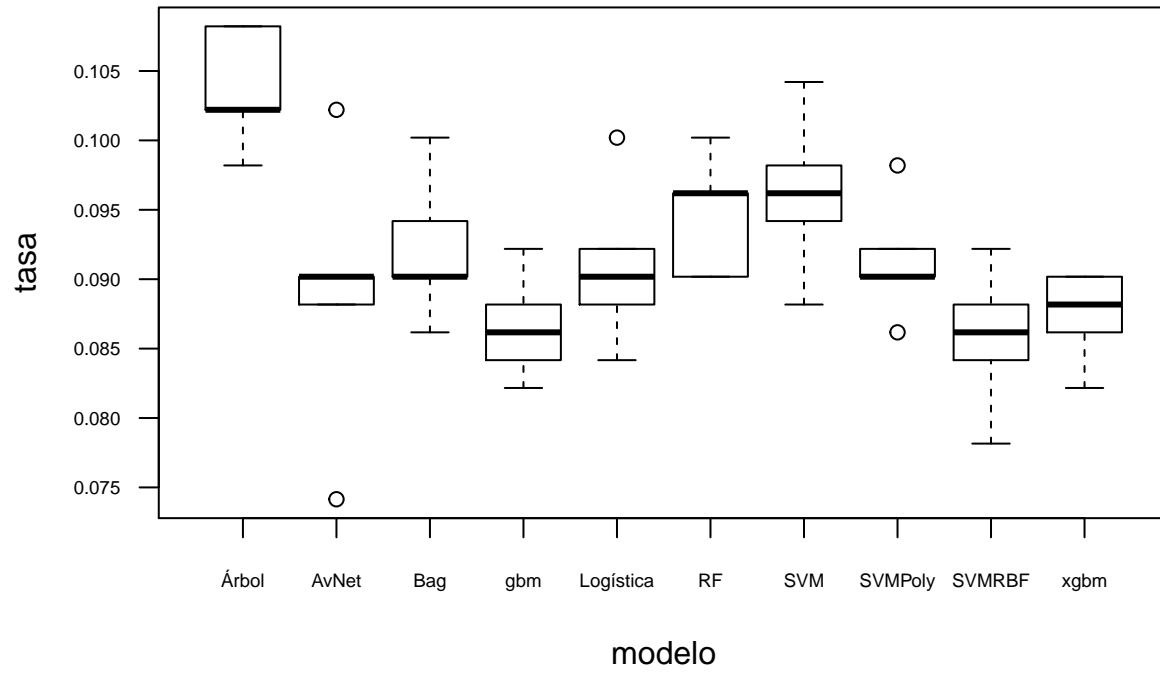
union<-rbind(logi,avnet,arbol,bag,rf, gbm, xgbm, svm, svmp, svmrbf)

par(cex.axis=0.6, cex=1, las=1)
boxplot(data=union,tasa~modelo,main="TASA FALLOS")

```



## TASA FALLOS



```
boxplot(data=union, auc~modelo, main="AUC")
```

## AUC

